1 Introduction

(1) With the introduction of `bool` (and its constants `true` and `false`) into the C++ language, it seems we have actually lost some of the usual operations available on boolean types. We used to write code like this:

```cpp
extern bool f(), g();
main()
{
 if (f() | g()) controlled_statement();
}
```

(2) The intent was clear: evaluate both `f()` and `g()` and execute the controlled statement if either one returned `true`. The same argument applies to the `&` and (perhaps more importantly) `^` operators.

(3) On the other hand, one might argue that it is not necessary to extend these bitwise operators to the `bool` type because their current definitions make the above code work just fine. That is, the values returned by `f()` and `g()` are converted to `int`, the bitwise operator is applied, and the result converted to `bool` for the `if` statement. I have the following objections to this logic:

- I would like compilers to warn about the automatic conversion of an `int` to a `bool`. However, I don’t want a warning in this context. Similarly, warnings about automatic conversion from `bool` to `int` are reasonable except in this context.
- It would be nice to be able to “correctly” overload on an expression like this.

(4) Similarly, the assignment operators should be defined for `bool`. Moreover, it would be nice to have assignment versions of the logical operators.

2 Define `&`, `|`, and `^` for `bool`

(1) Add the following to the beginning of the description of the bitwise and operator (5.11 `[expr.bit.and]`):

   If both operands are of type `bool`, the result is of type `bool` and is `true` only if both operands are `true`. Otherwise, ...

(2) Add the following to the beginning of the description of the bitwise exclusive or operator (5.12 `[expr.xor]`):

   If both operands are of type `bool`, the result is of type `bool` and is `true` only if the operands are unequal. Otherwise, ...

(3) Add the following to the beginning of the description of the bitwise or operator (5.11 `[expr.or]`):

   If both operands are of type `bool`, the result is of type `bool` and is `false` only if both operands are `false`. Otherwise, ...
3 Define &=, |=, and ^= for bool


   If \( E_1 \) is not of type bool, ...

(2) It then goes on to describe the rules we all know and love. Interestingly, it doesn’t say what is supposed to happen if \( E_1 \) is of type bool. I propose to remove the above quoted phrase from the text of the Working Paper.
1 Introduction

(1) With the introduction of `bool` (and its constants `true` and `false`) into the C++ language, it seems we have actually lost some of the usual operations available on boolean types. We used to write code like this:

```cpp
extern bool f(), g();
main()
{
    if (f() | g()) controlled_statement();
}
```

(2) The intent was clear: evaluate both `f()` and `g()` and execute the controlled statement if either one returned `true`. The same argument applies to the `&` and (perhaps more importantly) `^` operators.

(3) On the other hand, one might argue that it is not necessary to extend these bitwise operators to the `bool` type because their current definitions make the above code work just fine. That is, the values returned by `f()` and `g()` are converted to `int`, the bitwise operator is applied, and the result converted to `bool` for the `if` statement. I have the following objections to this logic:

   • I would like compilers to warn about the automatic conversion of an `int` to a `bool`. However, I don’t want a warning in this context. Similarly, warnings about automatic conversion from `bool` to `int` are reasonable except in this context.

   • It would be nice to be able to “correctly” overload on an expression like this.

(4) Similarly, the assignment operators should be defined for `bool`. Moreover, it would be nice to have assignment versions of the logical operators.

2 Define &, |, and ^ for bool

(1) Add the following to the beginning of the description of the bitwise and operator (5.11 [expr.bit.and]):

   If both operands are of type `bool`, the result is of type `bool` and is `true` only if both operands are `true`. Otherwise, ...

(2) Add the following to the beginning of the description of the bitwise exclusive or operator (5.12 [expr.xor]):

   If both operands are of type `bool`, the result is of type `bool` and is `true` only if the operands are unequal. Otherwise, ...

(3) Add the following to the beginning of the description of the bitwise or operator (5.11 [expr.or]):

   If both operands are of type `bool`, the result is of type `bool` and is `false` only if both operands are `false`. Otherwise, ...

---

WG21/N0432 Completing Boolean

Doc No: X3J16/94-0045
WG21/N0432
Date: May 31, 1994
Project: Programming Language C++
Reply-To: Neal M Gafter
neal@cs.rochester.edu

1 Introduction

(1) With the introduction of `bool` (and its constants `true` and `false`) into the C++ language, it seems we have actually lost some of the usual operations available on boolean types. We used to write code like this:

```cpp
extern bool f(), g();
main()
{
    if (f() | g()) controlled_statement();
}
```

(2) The intent was clear: evaluate both `f()` and `g()` and execute the controlled statement if either one returned `true`. The same argument applies to the `&` and (perhaps more importantly) `^` operators.

(3) On the other hand, one might argue that it is not necessary to extend these bitwise operators to the `bool` type because their current definitions make the above code work just fine. That is, the values returned by `f()` and `g()` are converted to `int`, the bitwise operator is applied, and the result converted to `bool` for the `if` statement. I have the following objections to this logic:

   • I would like compilers to warn about the automatic conversion of an `int` to a `bool`. However, I don’t want a warning in this context. Similarly, warnings about automatic conversion from `bool` to `int` are reasonable except in this context.

   • It would be nice to be able to “correctly” overload on an expression like this.

(4) Similarly, the assignment operators should be defined for `bool`. Moreover, it would be nice to have assignment versions of the logical operators.

2 Define &, |, and ^ for bool

(1) Add the following to the beginning of the description of the bitwise and operator (5.11 [expr.bit.and]):

   If both operands are of type `bool`, the result is of type `bool` and is `true` only if both operands are `true`. Otherwise, ...

(2) Add the following to the beginning of the description of the bitwise exclusive or operator (5.12 [expr.xor]):

   If both operands are of type `bool`, the result is of type `bool` and is `true` only if the operands are unequal. Otherwise, ...

(3) Add the following to the beginning of the description of the bitwise or operator (5.11 [expr.or]):

   If both operands are of type `bool`, the result is of type `bool` and is `false` only if both operands are `false`. Otherwise, ...
3 Define &=, |=, and ^= for bool


   If \( E_1 \) is not of type bool, ...

(2) It then goes on to describe the rules we all know and love. Interestingly, it doesn’t say what is supposed to happen if \( E_1 \) is of type bool. I propose to remove the above quoted phrase from the text of the Working Paper.
1 Introduction

(1) With the introduction of \texttt{bool} (and its constants \texttt{true} and \texttt{false}) into the C++ language, it seems we have actually lost some of the usual operations available on boolean types. We used to write code like this:

\begin{verbatim}
extern bool f(), g();
main()
{
    if (f() | g()) controlled_statement();
}
\end{verbatim}

(2) The intent was clear: evaluate both \texttt{f()} and \texttt{g()} and execute the controlled statement if either one returned \texttt{true}. The same argument applies to the \& and (perhaps more importantly) ^ operators.

(3) On the other hand, one might argue that it is not necessary to extend these bitwise operators to the \texttt{bool} type because their current definitions make the above code work just fine. That is, the values returned by \texttt{f()} and \texttt{g()} are converted to \texttt{int}, the bitwise operator is applied, and the result converted to \texttt{bool} for the \texttt{if} statement. I have the following objections to this logic:

- I would like compilers to warn about the automatic conversion of an \texttt{int} to a \texttt{bool}. However, I don’t want a warning in this context. Similarly, warnings about automatic conversion from \texttt{bool} to \texttt{int} are reasonable except in this context.
- It would be nice to be able to “correctly” overload on an expression like this.

(4) Similarly, the assignment operators should be defined for \texttt{bool}. Moreover, it would be nice to have assignment versions of the logical operators.

2 Define \&, |, and ^ for \texttt{bool}

(1) Add the following to the beginning of the description of the bitwise and operator (5.11 [expr.bit.and]):

If both operands are of type \texttt{bool}, the result is of type \texttt{bool} and is \texttt{true} only if both operands are \texttt{true}. Otherwise, ...

(2) Add the following to the beginning of the description of the bitwise exclusive or operator (5.12 [expr.xor]):

If both operands are of type \texttt{bool}, the result is of type \texttt{bool} and is \texttt{true} only if the operands are unequal. Otherwise, ...

(3) Add the following to the beginning of the description of the bitwise or operator (5.11 [expr.or]):

If both operands are of type \texttt{bool}, the result is of type \texttt{bool} and is \texttt{false} only if both operands are \texttt{false}. Otherwise, ...
3 Define &=, |=, and ^= for bool


   If E1 is not of type bool, ...

(2) It then goes on to describe the rules we all know and love. Interestingly, it doesn’t say what is supposed to happen if E1 is of type bool. I propose to remove the above quoted phrase from the text of the Working Paper.